形状, 箭头

描述已自动生成

**Panorama Image Construction**

**Based on Matlab**

**Name：**Junyou Chen

**Student ID：**202200171208

**Class：** Robot Class 22. 1

November 10 2024

**Abstract**

This project aims to develop a Matlab-based panorama stitching program that can automatically determine the optimal stitching order for a set of 20 images and seamlessly merge them into a single panoramic image. The core strategy involves first identifying and stitching the two most highly matched images from the target dataset to form a base panorama. Then, through an iterative process, the image with the highest match to the current panorama is selected for stitching, continuing until all images are integrated into the final seamless panorama.

Prior to processing, image preprocessing steps enhance the program’s adaptability and generalizability across different image sets. During feature recognition and matching, the VLFeat library is used to implement SIFT (Scale-Invariant Feature Transform) for extracting keypoints and descriptors. Precise feature matching between images is achieved by calculating Euclidean distances between feature descriptors. In the stitching phase, alignment transformation matrices are calculated to merge images, and Alpha Blending is used for smooth blending in overlapping areas to create natural transitions. Additionally, a user interface (UI) was designed to facilitate testing and user interaction, improving usability and interactivity.

Through testing, this project demonstrates the effectiveness of the developed Matlab-based panorama stitching program, capable of handling image sets of varying numbers, formats, and sizes while achieving basic seamless stitching. However, limitations were identified, including potential suboptimal stitching order, insufficient robustness to positional and rotational variations, limited image blending quality, low computational efficiency, and a simplistic unprocessed image selection strategy. These factors may affect the quality of the final panorama. Suggestions for future improvements are also provided.

**Key words: Panorama Image, Alpha Blending,** **SIFT，Image Feature Detection**

# PART 1 Task Description

## 1.1 Project Task

Create a MATLAB program to determine the correct stitching order of 20 images and stitch the image set into a visually seamless panorama, then output the result.

## 1.2 Image Information

A set of 20 images, named from "q1.jpg" to "q20.jpg," each with the same dimensions (600x400 pixels), the same format (.jpg), and in random order. Each image was taken from a different area of the same building facade.

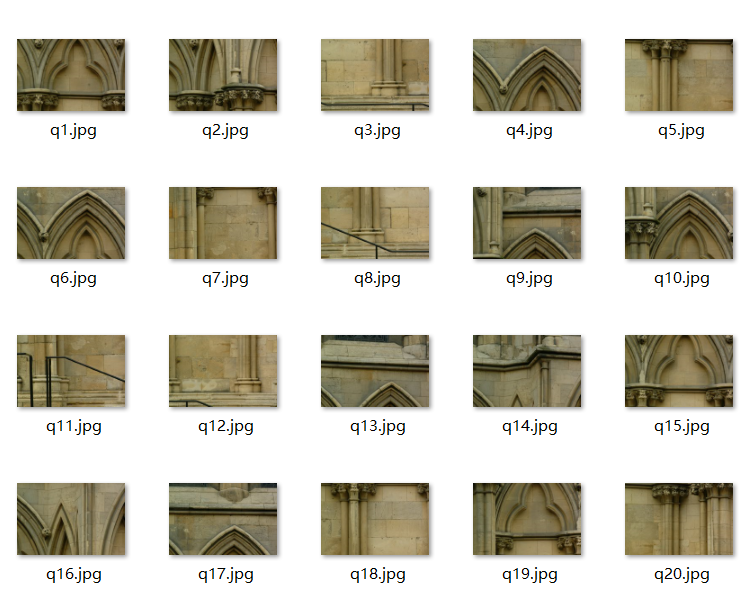
****

Fig 1 Project given set of images

# PART 2 Processing Scheme

## Project strategy-main function

Initially, the two images with the highest match score are selected to minimize stitching errors. Then, through a loop, the next unprocessed image with the best match to the current stitched image (the "main image") is chosen, ensuring that each step of the stitching process minimizes errors as much as possible. During the stitching process, any NaN values encountered will be replaced with 0 to prevent them from affecting subsequent stitching steps.

|  |
| --- |
| **Panorama Stitching Strategy:** |
| **Step 1: Initialization：**Load the VLFeat library (vlfeat-0.9.21) and use the loadImages function to load the image dataset.  **Step 2: Get Initial Images：**The getTargetInit function selects the two images with the highest match score to serve as the initial images for stitching.  **Step 3: Initial Stitching：**The stitching function stitches the initial two images together to generate a preliminary stitching result, which is then displayed. Any NaN values generated during the stitching process are replaced with 0.  **Step 4: Generate Updated Processed and Unprocessed Image Lists：**After each stitching, the processed images are added to the processedSet list, and the unprocessed images are removed from the unprocessedSet list.  **Step 5: Loop Stitching：**If there are more than two images, the loop stitching process begins. In each iteration, the function calculates the match score between the current stitched image and the unprocessed images, selecting the most suitable image for stitching. After each stitching, NaN values are replaced and the lists are updated.  **Step 6: Stitching Complete：**Once the loop ends, all images are stitched into a single panoramic image, and the final stitched result is displayed after removing the black border. |

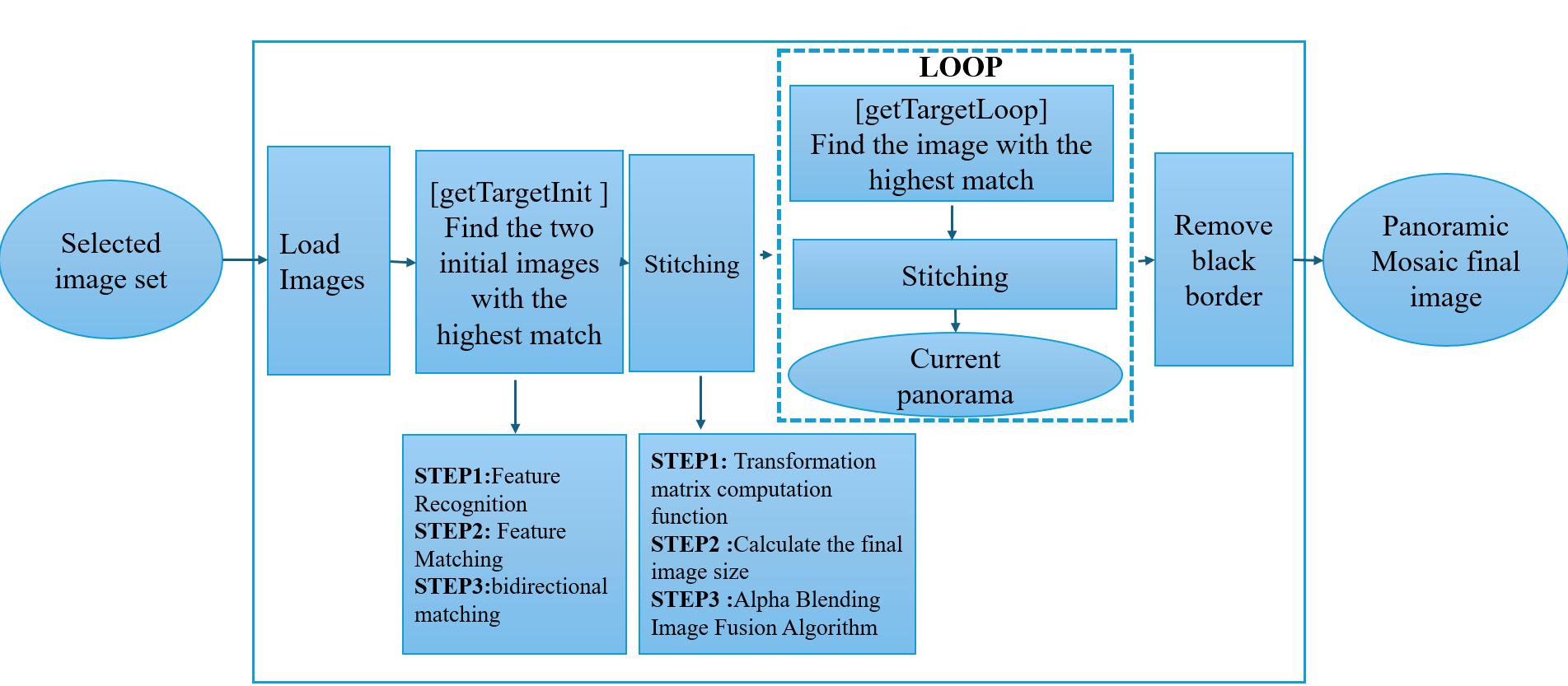


Fig 2 Strategy sketch

## Step 1: Image Set Loading

### 2.1.1 Code strategy

In the image acquisition strategy for the panoramic image stitching system, the process begins by allowing the user to select image files through a graphical user interface (GUI). Upon selecting the images, their file paths and names are stored in a structure. The images are then loaded into a cell array, `imgSet`, where each element corresponds to an individual image. If the images are in PNG format, they are temporarily converted to JPG format for consistent processing. The function `loadImages` handles the conversion and loading of images, ensuring that the system supports both PNG and JPG formats. The final cell array, containing the loaded images, is used for the stitching procedure. This approach ensures flexibility in handling different image formats and efficiently organizes the images for further processing.

### 2.1.1 Function introduction

#### Image Set Loading Function — loadImages

**Inputs:**

**imageFiles:** An array of structs containing information about the image files (e.g., file name and path).

**Outputs:**

**imgSet:** A cell array containing the loaded images.

**numImages:** An integer count of the total number of images processed.

**Function Logic:**

First, the number of image files, numImages, is obtained using `length(imageFiles)`. Then, an empty cell array, imgSet, is created to store the loaded images. The function proceeds by iterating through each image file in imageFiles:

* If the file is in PNG format, it is temporarily converted to JPG format, loaded, and the temporary JPG file is deleted afterward.
* If the file is in JPG or JPEG format, it is read and loaded directly.

Finally, the function outputs the cell array imgSet, which contains all the loaded images.

The main advantage of using a `cell` array to store images is its flexibility in holding data of varying sizes and types. Additionally, the flexibility of `cell` arrays allows for easy insertion, deletion, or access of images during the storage process, providing improved processing efficiency and scalability of the code.

## Step 2: Feature Recognition and Matching

### 2.2.1 Code strategy

In this algorithm, the project employs a feature-based image registration method using SIFT (Scale-Invariant Feature Transform). First, SIFT features and descriptors are extracted from all input images. A bidirectional matching strategy is then applied to compute the matching scores for each image pair, ensuring robustness in the matching results. For each image pair, matching is performed in both directions: from the first image to the second, and from the second image to the first. The matching score is determined by the sum of the matches from both directions, with higher scores indicating better matches. This approach effectively mitigates the risk of mismatches caused by one-way matching, enabling the selection of the optimal initial image pair for stitching. This strategy ensures the accuracy and stability of image registration, particularly when dealing with a large number of images, by reducing the impact of erroneous matches.

### 2.2.1 Function introduction

#### (1) Feature Recognition — getSIFTFeatures Function

**Input Parameters:**

**image:** The input image, which can be either grayscale or RGB.

**edgeThresh:** The threshold for SIFT edge detection, which adjusts the sensitivity of feature extraction.

**Output Parameters:**

**f:** The SIFT feature frames matrix. Each column represents the position and scale information of a feature point (e.g., position, scale, orientation, etc.).

**d:** The SIFT descriptor matrix. Each column corresponds to a descriptor of the feature point, a vector representing the local intensity distribution around the feature point, used for subsequent image matching.

**Function Logic:**

Grayscale images are more suitable for feature extraction, so the input color image is first converted to grayscale and then to single-precision format for compatibility with the SIFT algorithm. The vl\_sift function from the VLFeat library is used to extract the SIFT features and descriptors from the image. The vl\_sift function automatically detects keypoints in the image and computes a descriptor for each keypoint. The number and quality of the keypoints are closely related to the setting of the edgeThresh parameter; lower thresholds will detect more keypoints, while higher thresholds will reduce the number of keypoints detected.

The getOrderInit function performs feature recognition and matching on the target image set, outputting the two images with the highest matching degree for the initial panorama stitching. The getOrderLoop function performs feature recognition and matching between the existing panorama and the unmatched image set, outputting the image with the highest matching degree for further panorama stitching.

**Theory of vl\_sift (SIFT Algorithm):**

The vl\_sift function in the VLFeat library is used to extract SIFT (Scale-Invariant Feature Transform) features. SIFT is used to detect keypoints in an image that are invariant to scale and rotation, and compute a descriptor for each keypoint.

In SIFT, feature points are detected at different scales (i.e., blurred images of various sizes), so a Gaussian pyramid is constructed with images at different blur levels. A Difference of Gaussian (DoG) image is computed to find extrema in the scale-space. Then, a quadratic surface is fitted to precisely locate each potential keypoint, filtering out points with low contrast and strong edge responses. Next, a dominant orientation is assigned to each keypoint to make the feature descriptor rotation-invariant. The feature descriptor is computed using the gradient information from the local region of the keypoint, which is invariant to scale, rotation, and minor changes in illumination. Finally, the positions, scales, orientations, and descriptors of all keypoints are returned.

#### (2) Feature Matching — getMatches Function

**Input Parameters:**

**f1:** Feature point position matrix for the first image, with size 3 x N, where N is the number of feature points. Each column represents the position of a feature point, containing three elements: x-coordinate, y-coordinate, and scale.

**d1:** Feature descriptor matrix for the first image, with size D x N, where D is the dimension of each feature descriptor and N is the number of feature points. Each column is a descriptor for a feature point.

**f2:** Feature point position matrix for the second image, with size 3 x M, similar to f1, containing M feature points' positions.

**d2:** Feature descriptor matrix for the second image, with size D x M, similar to d1, containing M feature points' descriptors.

**Output Parameters:**

**potential\_matches:** A 3D matrix of size numMatches x 3 x 2, where numMatches is the number of matching point pairs. Each match contains the feature point positions (x, y) and corresponding scale information from both images. The first layer stores the coordinates and scale of the matched points from the first image, and the second layer stores the coordinates and scale of the matched points from the second image.

**scores:** A vector of matching scores, with length numMatches. Each score corresponds to a match pair, where higher scores indicate better matching quality.

**Function Logic:**

The function uses vl\_ubcmatch from the VLFeat library to match the feature descriptors d1 and d2 between two images. vl\_ubcmatch finds the most similar descriptor pairs by computing the Euclidean distance between descriptors and returns the matching indices along with the scores for each match. The output contains descriptor index pairs (i.e., indices of the matched descriptors), which point to the descriptors in d1 and d2. These descriptor indices are then mapped back to their corresponding feature point positions in f1 and f2, and the feature points' positions and scales are stored in the pairs array. Each row of the pairs array represents a matching pair, containing the feature point positions (x, y) and scale information for both images. For consistency, the position and scale information is filled into two layers of the pairs array: the first layer for the feature points from the first image, and the second layer for the feature points from the second image.

**Theory of vl\_ubcmatch Function:**

The vl\_ubcmatch function matches two sets of descriptors by comparing their distances and finding the most similar descriptors in each set.

First, the Euclidean distance (L2 norm) between two descriptor sets (d1 and d2) is computed. For each descriptor in d1 (denoted as d1\_i), the most similar descriptor in d2 (denoted as d2\_j) is found based on the minimum distance, and this pair is considered the best match. To reduce false matches, Lowe’s ratio test is applied to validate the reliability of each match. Specifically, for d1\_i and d2\_j, if the distance to the nearest neighbor (best match) is dist1, and the distance to the second nearest neighbor is dist2, the ratio test checks if the ratio of dist1 to dist2 is smaller than a threshold to confirm the match's validity. The function then outputs the matching indices and the corresponding scores.

The vl\_ubcmatch function performs unidirectional matching, typically producing more precise results but may miss some matches. Further functions are used later to correct these missed matches.

#### (3) Matched Image Output Functions

**Functions**: getTargetInit & getTargetLoop

**Function Description:**

The getTargetInit function is used to begin the image set stitching process. It performs feature recognition and matching on the target dataset, outputting the two images with the highest matching score for subsequent stitching. The getTargetLoop function is used in the iterative stitching process. It performs feature recognition on the unprocessed image set and matches it with the previous panorama image (referred to as the master image) for subsequent stitching.

**Function** **Logic:**

The getTargetInit function loops through the target dataset and extracts the SIFT features of each image using the getSIFTFeatures function. Then, it calculates the bidirectional matching scores between pairs of images using the getMatches function in a nested loop. The bidirectional matching involves calculating the matching score from the master image to the target image and the matching score from the target image to the master image. The purpose of bidirectional matching is to reduce the risk of false matches. Finally, the two images with the highest matching scores are selected and their indices are output.

The basic logic of the getTargetLoop function is the same as getTargetInit, except that it uses a single loop to find the image in the unprocessed image set that has the highest matching score with the master image.

## Step 3: Panorama Stitching of Two Images

### 2.3.1 Code strategy

In the panorama stitching algorithm, the project combines two images using a transformation matrix computed through feature matching and RANSAC. The transformation matrix aligns the images while accounting for their relative positions. We avoid cylindrical projection for transforming the images, as multiple stitchings can lead to significant image distortion. Instead, the algorithm calculates the cumulative transformation matrix directly from the initial image alignment and adjusts it to accommodate the boundaries of the stitched panorama. To ensure a seamless blend, the images are combined using alpha blending, which carefully merges them by adjusting pixel intensity based on overlapping regions. The final panorama is obtained by calculating the new image dimensions, applying the transformations, and ensuring smooth transitions between the two images. This method prioritizes robustness in alignment while minimizing distortion in the final result.

### 2.3.2 Function introduction

#### (1) Stitching Function — stitching

**Input Parameters:**

**img1:** The first image, which has already been transformed through cylindrical projection.

**img2:** The second image to be stitched.

**Output Parameters:**

**newImg:** The final stitched image, resulting in a panoramic view.

**Supporting Functions：**

**computeBoundingBox：**Calculates the minimum boundary of the Mosaic image to determine the size of the panorama

**adjustTranslation：**Adjust the translation matrix to align the images correctly in the panoramic coordinate system.

**Function Logic:**

First, the alignment transformation matrix for the input images is calculated to align the position of image 2 with image 1. Next, the cumulative transformation matrix absoluteTrans is initialized, and the transformation matrices of both images are combined to achieve the final cumulative transformation. Using this cumulative transformation matrix, the function then calculates the dimensions of the resulting panorama to accommodate the full content of both images. By comparing the transformed coordinates, the final image’s width and height are updated, and the four boundaries (minX, maxX, minY, maxY) of the stitched image are determined. Afterward, the translation component of the transformation matrix is updated to ensure proper alignment within the new panoramic coordinate system. Finally, the merge function is used to blend the two images into the new panorama, aligning them with the adjusted transformation matrix.

#### (2) Transformation matrix computation function—computeMatrix

**Input Parameters:**

**img1:** The first input image.

**img2:** The second input image.

**Output Parameters:**

**T:** A 3x3x2 matrix where T(:,:,1) is the identity matrix and T(:,:,2) is the transformation matrix computed using RANSAC.

**Function Logic:**

First, the RANSAC parameters are initialized. Secondly, SIFT features and descriptors of the two images were extracted by getSIFTFeatures. The function then computes the transformation matrix T through the RANSAC algorithm and returns the result.

**RANSAC algorithm logic：**

RANSAC (Random Sample Consensus) is an iterative algorithm used to estimate model parameters from data that may contain outliers.

The process begins by randomly selecting a minimal sample set from the data to compute a candidate model. Next, the algorithm calculates the fitting error for all data points using this model and classifies the points as inliers or outliers. The model's quality is then assessed; if the number of inliers is sufficiently large, it indicates that the model is accurate, and the optimal model is updated. This process is repeated multiple times, with a new random sample set selected each time to compute a new candidate model. Finally, the algorithm returns the model with the most inliers as the final result.

#### (3) Alpha Blending Image Fusion Algorithm — AlphaBlending

**Input Parameters:**

**img1:** The first input image (RGB or grayscale).

**img2:** The second input image (RGB or grayscale).

**transforms:** A 3x3xN transformation matrix, where each 3x3 matrix represents the transformation information for the corresponding image.

**newHeight:** The height of the resulting panoramic image.

**newWidth:** The width of the resulting panoramic image.

**Output Parameters:**

**newImg:** The synthesized panoramic image.

**Function Logic:**

Alpha Blending, also known as weighted average blending, is a technique commonly used in image stitching and fusion. It applies weighted averaging to the pixel values in overlapping areas to reduce visible seams at the edges, resulting in a smooth and natural fusion of images.

First, both input images are converted to double precision to ensure computational accuracy, and transparency masks are created for each image for subsequent transparency blending. Next, a 3D transparency mask filled with ones (mask) is created for each channel of both images, which will be used in the blending process. Then, the function iterates through the input images, applies the transformation matrices (transforms), and calculates the position of each image to define the boundaries of the final panoramic image dimensions. A composite image and denominator matrix are initialized, where the denominator matrix records the accumulation count of each pixel location to facilitate the normalization in the transparency blending. Each image is then positioned and overlaid onto the panorama using the transformation matrices, and the denominator matrix is updated to reflect transparency blending. Finally, the pixel values of the composite image are divided by the accumulation count to achieve transparency normalization, creating a seamless and naturally blended panorama.

#### (4) Remove the black boundary function — removeBlackBorder

**Input Parameters:**

**inputImage:** The input image, which can be a grayscale, RGB, uint8, or double image.

**threshold:**A value that defines the black color threshold, typically between 0 and 255.

**Output Parameters:**

**outputImage:** The cropped image with black borders removed.

**Function Logic:**

First, convert the image to a grayscale uint8 type, which simplifies the process of identifying black areas (as black areas appear as low brightness in grayscale images). Next, convert the grayscale image to a binary image, where pixels below a certain threshold are marked as 1 (representing black areas), and other pixels are marked as 0 (non-black areas). Then, based on the coordinates of the non-black areas, determine the rectangular region containing the valid content. Finally, crop the image based on the calculated boundaries and output the image with the black borders removed.

## Step 4: UI Interface Design

The design of the user interface (UI) for the Panoramic image building system follows a straightforward layout, enabling users to interact efficiently with the image stitching process. The interface is designed to provide clarity, ease of use, and real-time feedback throughout the image selection, stitching, and saving stages.

### 5.1 UI Components:

**1. Main Window:**

The main window is a figure with the title "Panoramic image building system", sized 800x600 pixels. This window houses all the UI controls and displays the panorama image once stitching is complete.

**2. Buttons:**

* **Select Images**: Opens a file explorer for users to select the input image set.
* **Start**: Begins the image stitching process after the images have been selected.
* **Save**: Allows users to save the resulting panorama image once stitching is completed. This button is initially disabled and only enabled after the stitching process is finished.
* **Reset**: Resets the system, clearing selected images and final output, allowing users to select a new set of images.
* **Exit**: Closes the UI and exits the system.

**3. Progress Bar:**

A text field serves as a progress bar, providing feedback to the user about the current status of the stitching process (e.g., progress percentage or "To be started" before any action). This dynamic feedback helps users track the progress in real-time.

**4. Display Area:**

The central image area is an axes object, which displays the panorama image as it is being stitched together. Initially, this area is empty but updates continuously as images are processed.

### 5.2 Subfunction Introduction

#### (1) Stitching Function: main

**Function Logic:**

See "Step 3: Panorama Stitching of the Image Set."

Once the stitching process begins, all buttons except the "Exit" button are disabled to prevent interruption of the stitching process. After the stitching is completed, the buttons are re-enabled.

#### (2) File Selection Function: selectFiles

**Function Logic:**

**- Step 1:** Open the file selector, allowing the user to select multiple image files (supports multi-selection).

**- Step 2:** Store the selected file paths and names in `imageFiles`.

**- Step 3:** If selection is successful, pop up a confirmation message box; if no files are selected, pop up an error message.

#### (3). Progress Bar Update Function: updateProgress

**Function Logic:**

**- Step 1:** Calculate the percentage based on the current progress and total progress.

**- Step 2:** Update the progress bar text to display the stitching progress in real-time.

#### (4) File Save Function: saveImage

**Function Logic:**

**- Step 1:** Check if the stitching is complete; if not, pop up a message indicating the image is not ready.

**- Step 2:** Open the file save dialog, allowing the user to select the save path and file name.

**- Step 3:** Save the final stitched image to the selected path and pop up a success message.

#### (5) System Reset Function: resetSystem

**Function Logic:**

**- Step 1:** Clear image files and stitching results (reset system state).

**- Step 2:** Enable the image selection button and disable other stitching-related buttons.

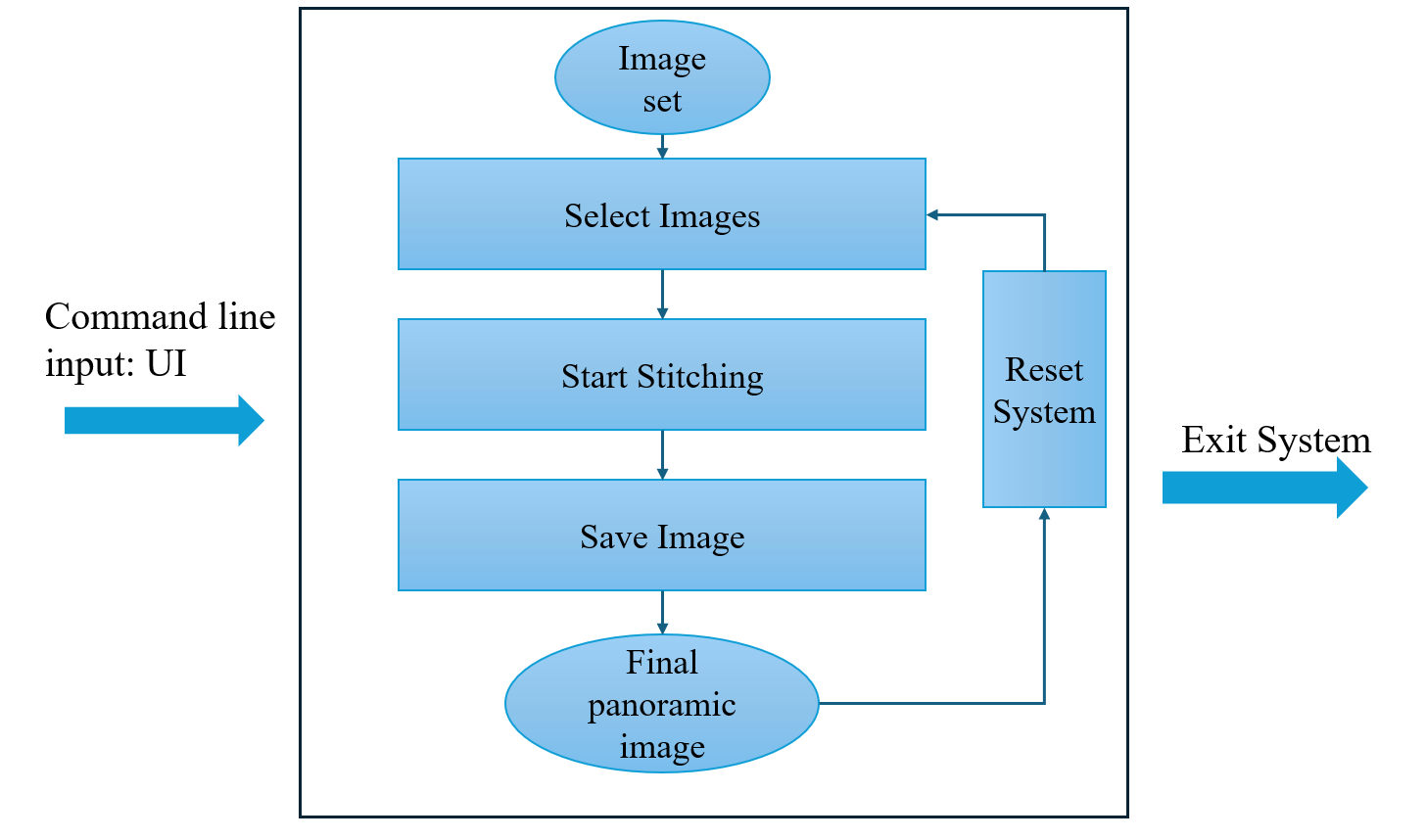
**- Step 3:** Clear the display area (`imgAxes`) to prepare for a new operation.

**- Step 4:** Update the progress bar to show the initial state.

**(6) Exit System Function: exitSystem**

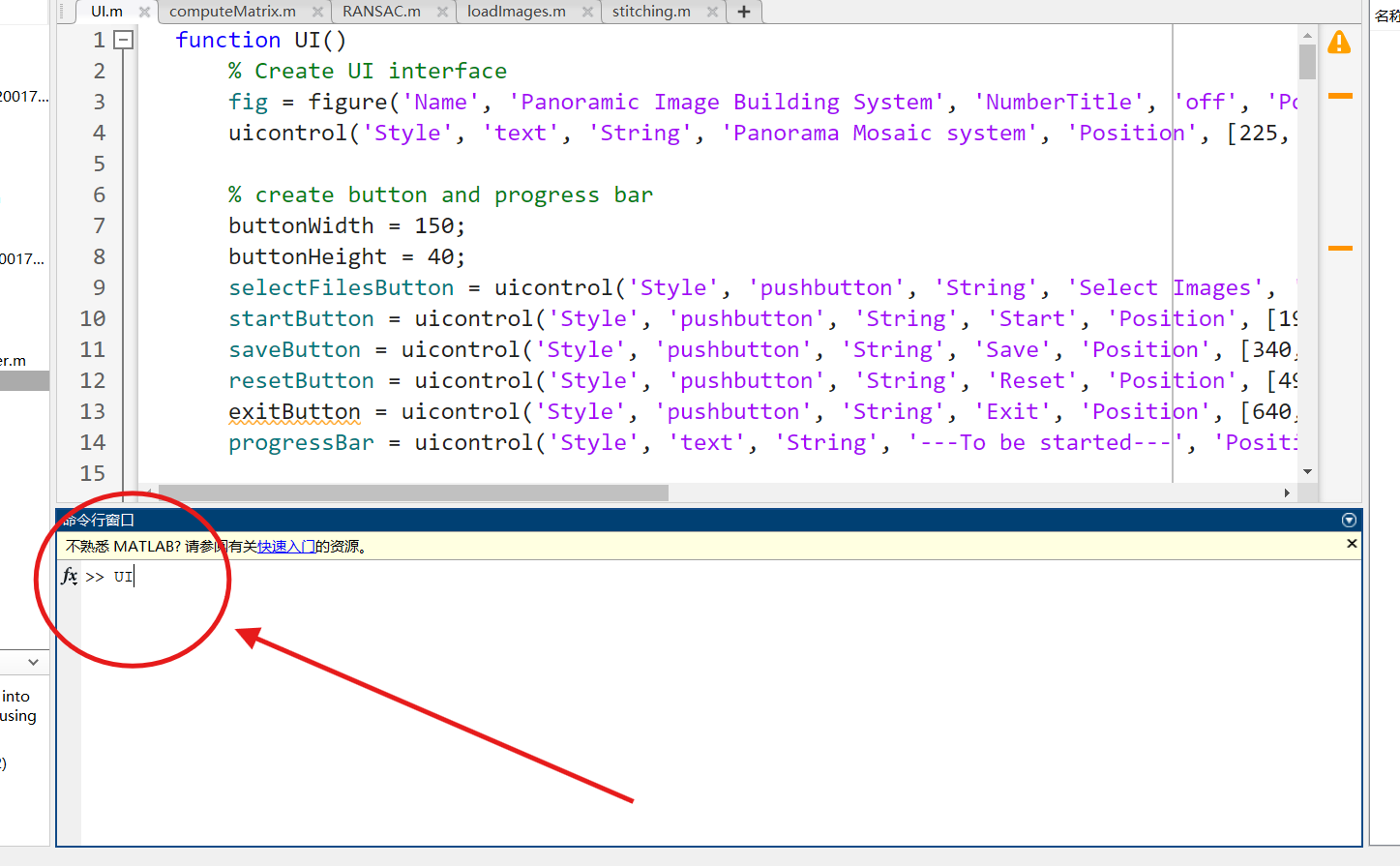
**Function Logic:** Close the main interface window and exit the program.

# PART 3 Process description



## 3.1 Enter the System

Open the code in MATLAB and type "UI" in the command line to enter the Panoramic image building system's UI interface.



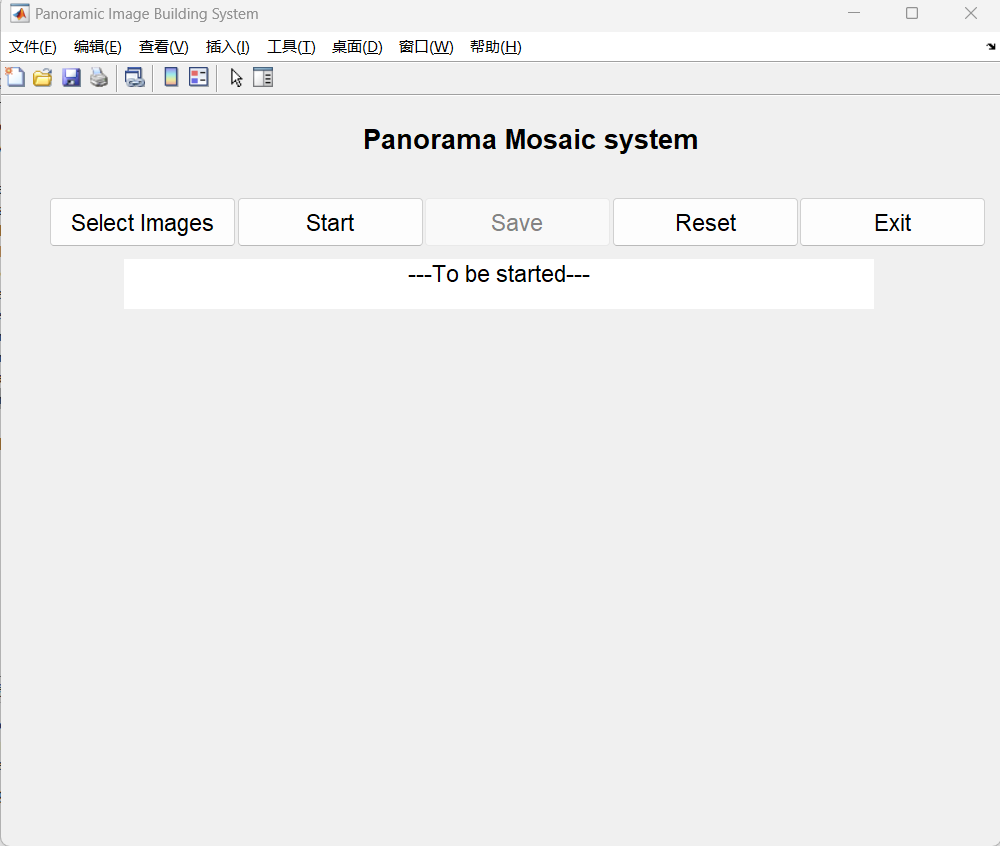
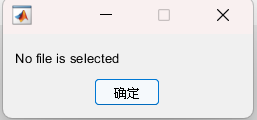


Fig 3 Enter the system procedure

## 3.2 Select Images

After clicking "Select Images," the file explorer will open, allowing you to select images from any path, in any quantity, and of any size (supports .jpg and .png formats). Once the selection is complete, a pop-up window will display "The image file is selected successfully!" to indicate a successful selection; if the pop-up shows "No file is selected," the selection has failed.

图形用户界面, 应用程序, PowerPoint

描述已自动生成 图形用户界面, 文本, 应用程序

描述已自动生成

Fig 4 Select Images

## 3.3 Start Stitching

Click the "Start" button to begin stitching the selected image set into a panorama. A progress bar in the center of the UI will display the stitching progress, and the window below will show the current stitching status. To ensure the system operates smoothly, all button functionalities except "Exit" will be disabled during the process.

图形用户界面

描述已自动生成

Fig 5 Start Stitching

## 3.4 Stitching Complete

Once the stitching is complete, the progress bar will display the final stitching order, and the window below will show the final panorama image.

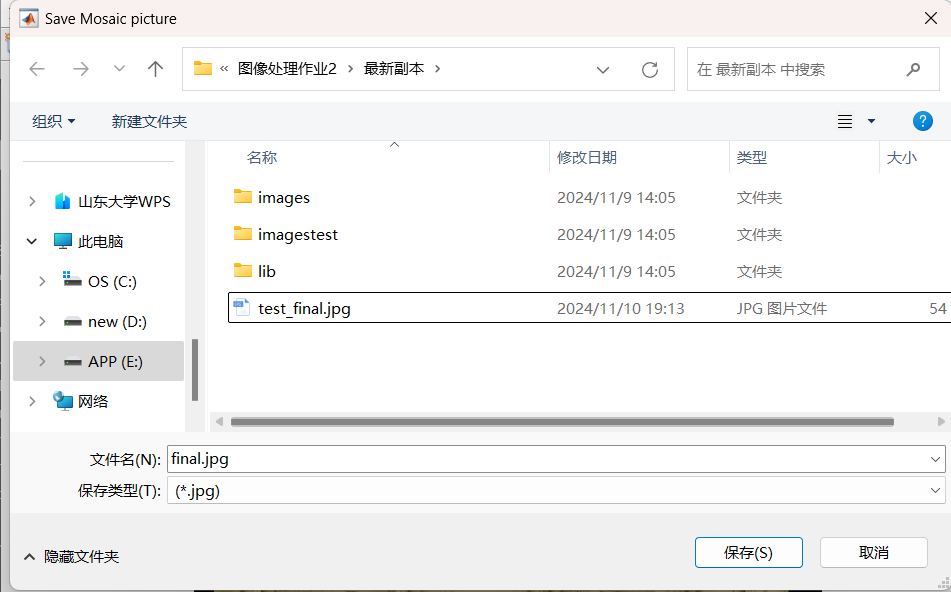
图片包含 图形用户界面

描述已自动生成

Fig 6 Stitching Complete

## 3.5 Save Image

After the stitching is complete, you can click the "Save" button to save the final image. The file explorer will open, allowing you to select the target save path and customize the file name. Once the image is saved successfully, the system will display a pop-up window with the message "Image saved successfully!"; if the pop-up shows "The save path and file name are not selected," the save operation has failed, and you will need to choose the correct path and file name.



图形用户界面, 应用程序

描述已自动生成

Fig 7 Save Image

## 3.6 Reset System

After completing one round of panorama stitching, you can click the "Reset" button to reset the system for a new round of image set stitching. A system pop-up will display "The system has been reset, please re-select the picture" to indicate that the reset was successful.

图形用户界面, Word

描述已自动生成

Fig 8 Reset System

# PART4 Result Presentation

## 4.1 Task Image Set

**1. Input:** The original dataset `images` consists of 20 images, each with a size of 600x400 pixels and in .jpg format. The images are unordered.

**2. Results:**



Fig 9 The final panoramic image of the task image set

**3. Final Stitching Order:**

【4->13->6->17->9->10->14->15->1->2->20->5->18->7->11->12】

**4. Conclusion:**

The algorithm in this project can generally achieve seamless panorama stitching.

## 4.2 Custom Image Set

**1. Input:** The custom dataset `imagestest` contains 9 images with varying sizes, in .png format, and unordered.

**2. Results vs Original Images:**

教堂的建筑

描述已自动生成

Fig 10 The final panoramic image of the self-built image set (left) compared to the original image (right)

**3. Final Stitching Order:** 【5->7->6->8->1->2->4->9->3】

**4. Conclusion:** The algorithm in this project can generally achieve panorama stitching for any number of images, in any format, and with any size.

**PART 5 Project Defect**

## 5.1 Disadvantages:

**1.Suboptimal Stitching Order:**

The current stitching order is based on selecting the image with the highest match score at each step, which is a greedy strategy. This approach may not result in the most optimal stitching order. For example, some images might be chosen early on, but they may not be the most suitable for the final sequence, causing significant overlap and distortion in the stitched result.

**2.Poor Robustness to Image Position and Rotation:**

The current method uses a straightforward matching and stitching strategy, which is not very robust to image transformations like rotation and translation. While feature matching techniques like SIFT are used, the geometric transformations between images (e.g., differences in rotation angles) may not be effectively handled, leading to stitching errors or gaps.

**3.Insufficient Image Blending:**

During stitching, the image blending relies on simple pixel overlay methods (e.g., alpha blending), but more advanced blending techniques (e.g., gradient blending or multi-scale blending) are not employed. This can result in noticeable seams, especially when there are differences in brightness or color between the images.

**4.Low Computational Efficiency:**

At each step, the method selects the image with the highest match score for stitching, which can lead to unnecessary repeated calculations and redundant steps. For large-scale image stitching, this approach can be computationally expensive and inefficient.

**5.Simple Strategy for Choosing Unprocessed Images:**

The strategy for selecting unprocessed images is to choose the one with the highest match score at each step. Although this approach is intuitive, it does not consider the global relationships between images. More complex image sorting or clustering methods are not used, which can lead to unnatural stitching results, especially when there is little overlap between images or fewer feature points.

## 5.2 Suggestions for Improvement:

1. Use image transformations like homography matrix estimation to address issues with rotation, translation, and perspective distortion between images.

2. Implement advanced blending techniques such as gradient or multi-scale blending to reduce the visibility of seams and create more natural-looking results.

3. Use more efficient algorithms, such as multi-threading or optimized feature matching methods, to reduce redundant computations.

4. Adopt global optimization techniques (e.g., constructing an image stitching graph and using shortest path algorithms) to determine the stitching order, resulting in a more ideal panorama.

# PART 6 Experimental Insights

Through this semester's Image Processing and Computer Vision course and the final project, I have gained a deep appreciation for the importance of combining theory with practice. In developing a Matlab-based panorama stitching program, I not only strengthened my understanding of theoretical knowledge from class, such as feature extraction, image stitching, and the RANSAC algorithm, but also learned how to apply these concepts to real-world problems and overcome specific technical challenges.

I realized that, while theoretical algorithms may be mathematically sound, they often require adjustments and optimizations for specific datasets and environments in practical applications. For instance, my initial inexperience with parameter settings for SIFT feature extraction led to lower accuracy in feature matching. Through continuous experimentation and adjustment, I gradually understood the impact of parameter selection on outcomes and learned how to optimize based on actual needs. Additionally, I gained insights into balancing algorithm efficiency and performance. In the early stages of the project, I focused more on functionality, overlooking code optimization, which resulted in slower program performance. As the project progressed, I began to learn how to improve efficiency through algorithm refinements and code optimizations.

Overall, this project has not only enhanced my technical skills but also strengthened my problem-solving abilities and innovative thinking. I look forward to applying these valuable lessons in future studies and work. Finally, I would like to thank Professor Song Ran for his guidance and support, which led me in my initial exploration and growth in the field of image processing and computer vision.

# Reference：

* https://github.com/joyeecheung/panoramic-image-stitching
* https://github.com/yrlu/image\_mosaic\_stitching
* https://github.com/BrandonHanx/AutoPanorama
* https://github.com/khushhallchandra/Automatic-Panoramic-Stitching